САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

КАФЕДРА МАТЕМАТИЧЕСКОГО МОДЕЛИРОВАНИЯ ЭНЕРГЕТИЧЕСКИХ СИСТЕМ

Ивашечкин Александр Павлович

Магистерская диссертация

Динамическая адаптация метода имитации отжига для решения задачи коммивояжера

Направление 01.04.02

Процессы управления

Магистерская программа Математическое и информационное обеспечение экономической деятельности

Научный руководитель,
кандидат физ. мат. наук,
доцент
Власова Т. В.

Санкт-Петербург

2018

Содержание

Введение

На сегодняшний день задача поиска кратчайшего пути между двумя пунктами является весьма востребованной: объем рынка транспортно-логистических услуг растет с каждым днем. Их главная цель — построение наиболее точного маршрута для обслуживания максимального количества клиентов. При этом очень важно учитывать, что выбор неудачного пути повлечет за собой дополнительные издержки.

Однако существует набор факторов, характеризующий продолжительность прохождения пути: время суток, погодные условия, а также «нагруженность» прохождения того или иного участка. Причем необходимо учитывать их взаимосвязь. Например, участок улицы между двумя перекрестками в разное время суток имеет разную загрузку: большую в час пик, а значит, длительность его прохождения отличается от ночного; погодные условия: туман, дождь – все это влияет на время прохождения участка.

Задача коммивояжера или TSP (Travelling Salesman Problem) – задача математического программирования, имеющая перед собой цель определить оптимальный маршрут движения торговца, которому необходимо побывать во всех пунктах, записанных в задании, за минимальное время и с наименьшими затратами. Аналогом для теории графов является поиск пути между двумя и более узлами с использованием критерия оптимальности [1].

Алгоритмы, позволяющие решить данную проблему, можно разделить на точные и эвристические. Для небольших задач (например, в целях первичного проектирования транспортной сети малых размеров) целесообразно будет использовать точные алгоритмы, так как для их реализации необходимы высокие вычислительные мощности, чего нельзя сказать о реальных задачах: как правило, для них необходимо использовать эвристические алгоритмы.

TSP является типичной задачей оптимизации, которая широко применяется при разработке программного обеспечения. Задача о коммивояжере представляет собой упрощенную модель для многих других задач дискретной

оптимизации, а также часто является подзадачей. Она служит катализатором, стимулирующим разработку наиболее эффективных методов, алгоритмов и способов их машинной реализации.

Задача коммивояжера формулируется очень просто: на плоскости (в пространстве) расположены N городов, заданы расстояния между каждой парой городов. Необходимо отыскать маршрут минимальной длины, посетив каждый города единожды и вернувшись в исходную точку. Однако стоит отметить, что это определение не учитывает возможность хаотичного изменения начальных данных задачи в процессе прохождения маршрута коммивояжером.

Обзор литературы

Первые вариации метода отжига были разработаны в 1953 году, когда Н. Метрополис вывел алгоритм симуляции формирования теплового равновесия в системе со многими степенями свободы при заданной температуре. В начале восьмидесятых годов. С.Киркпатрик впервые предложил использовать этот метод для решения оптимизационных задач, а не только для моделирования физических систем. История развития этого алгоритма и его модификаций подробно рассмотрена авторами Vasek Chvatal, William J. Cook, George B. Dantzig в работе [7].

Борознов В.О. и Гараба И.В. в статьях [2][3] поясняли, что алгоритм имитации отжига успешно применяется для широкого круга задач, в том числе и для решения проблемы TSP и приводили сравнительный анализ различных подходов к оптимизации задачи коммивояжера. В [6] были сформулированы общие принципы применения алгоритма имитации отжига для TSP: определение пространства решений, на котором идет поиск; задание структуры окрестности каждого решения; определение целевой функции алгоритма. Также было показано, что существует теоретическая возможность доказательства асимптотической сходимость алгоритма. В работе [9] авторами

Christos H. Papadimitriou и Kenneth Steiglitz приведены экспериментальные подтверждения эффективности алгоритма имитации отжига для задачи TSP. В статье [10] Носкова Е. предлагает усовершенствованный подход к построению алгоритма, с использованием эвристических подходов. В классическом алгоритме имитации отжига новое решение выбирается случайным образом, но можно использовать эвристики, исключающие полностью вероятностный характер выбора, направляющие поиск. В. П. Сигорский [12], Е. В. Маркова, А. Н. Лисенков [14] в качестве таких эвристик предлагали использовать построение расписаний по критическому пути, топологическое перемещение заданий, балансировка нагрузки на процессоры [10], оптимизация отдельных обособленных подграфов. О. Ope в [11] описывал применения алгоритма имитации отжига к решению задач оптимизации при вводимых ограничениях аппаратных ресурсов, но без ограничений на надежность, размышляя о необходимости динамической адаптации алгоритмов подобной группы для маршрутизации транспорта на больших сетях

Идеи динамической адаптации подробно рассмотрели Захаров В.В. и Мугайских А.В. в статье [23], где было приведено описание процедуры адаптации алгоритма применительно к TSP и продемонстрирована методика использования свойства временной состоятельности эвристических алгоритмов с целью повышения их эффективности.

Постановка задачи TSP

Задача коммивояжера – комбинаторная задача, для решения которой может применяться метод математического программирования. В современном виде она была сформулирована в 1934 году. В области оптимизации дискретных задач эта проблема часто используется для проверки и обкатки появляющихся решений, является полигоном, на котором испытываются все новые методы.

Чтобы привести задачу к общему виду, введем некоторые термины. Города перенумерованы числами $j \in T = (1,2,3,..,n)$. Маршрут коммивояжера может быть описан циклической перестановкой $t = (j_1, j_2, .., j_n, j_1)$, причем все $j_1..j_n$ – разные номера; повторяющийся в начале и в конце j_1, показывает, что перестановка зациклена [3]. Расстояния между парами вершин C_{ij} образуют матрицу C. Задача состоит в том, чтобы найти такой маршрут t, который имеет минимальную длину.

Относительно математической формулировки задачи коммивояжера нужно сделать два замечания.

Во-первых, в постановке C_{ij} означает расстояния, поэтому они должны быть неотрицательными, т.е. для всех $j \in T$:

$$C_{ij} \geq 0 \qquad (1)$$

$$C_{ij} \neq \infty \qquad (2)$$

(условие (2) означает запрет на петли в туре), симметричными, т.е. для всех i, j:

$$C_{ij} = C_{ji} \qquad (3)$$

и удовлетворять неравенству треугольника, т.е. для всех:

$$C_{ij} + C_{jk} \geq C_{ik} \qquad (4)$$

В постановке задачи говорится о произвольной матрице. Сделано это потому, что имеется много прикладных задач, которые описываются основной моделью, но всем условиям (2) - (4) не удовлетворяют. Особенно часто нарушается условие (3) (например, если значение – не расстояние, а плата за проезд: часто билеты туда и обратно стоят по-разному). Поэтому будем различать

два варианта задачи коммивояжера: симметричную задачу, когда условие (3) выполнено, и несимметричную - в противном случае. Условия (2) - (4) по умолчанию будем считать выполненными.

Второе замечание касается числа всех возможных вариантов. В несимметричной задаче коммивояжера все маршруты $t=(j_1, j_2,..., j_n, j_1)$ и $t=(j_1, j_n,..., j_2, j_1)$ имеют разную длину и должны учитываться оба. Разных вариантов очевидно $(n-1)!$.

Зафиксируем на первом и последнем месте в циклической перестановке номер j_1, а оставшиеся $n-1$ номеров переставим всеми $(n-1)!$ возможными способами. В результате получим все несимметричные маршруты. Симметричных маршрутов имеется в два раза меньше, т.к. каждый засчитан два раза: как t и как t', где t' - маршруты, симметричные t.

Допустим, что C состоит только из единиц и нулей. Тогда C можно интерпретировать как граф, где ребро (i, j) проведено, если $C_{ij}=0$, и не проведено, если $C_{ij}=1$. Тогда, если существует маршрут длины 0, то он пройдет по циклу, который включает все вершины по одному разу. Такой цикл называется *гамильтоновым циклом* [5]. Незамкнутый гамильтонов цикл называется гамильтоновой цепью (гамильтоновым путем).

В терминах теории графов симметричную задачу коммивояжера можно сформулировать так:

Дана полная сеть с n вершинами, длина ребра $(i, j)=C_{ij}$. Найти гамильтонов цикл минимальной длины.

В несимметричной задаче коммивояжера вместо “цикл” надо говорить “контур”, а вместо “ребра” - “дуги” или “стрелки”.

Некоторые прикладные задачи формулируются как задача коммивояжера, но в них нужно минимизировать длину не гамильтонова цикла, а гамильтоновой цепи. Такие задачи называются незамкнутыми. Некоторые модели сводятся к задаче о нескольких коммивояжерах, но изучение подобных случаев не входит в рамки этой дипломной работы.

Математическая модель задачи TSP

Пусть $I = \{0,...,n-1\}$ - множество индексов вершин из нашей задачи. Целевая функция f - суммарное расстояние маршрута, проходящего через все вершины графа рассматриваемой задачи.

Параметрами задачи являются элементы матрицы веса $C = \{c_{ij}, \forall i, j \in I\}$. Будем рассматривать задачи только с симметричной матрицей расстояний.

Переменными задачи являются элементы бинарной матрицы переходов между вершинами $X = \{x_{ij}, \forall i, j \in I\}$, которые равны 1, если в построенном маршруте для тестовой задачи присутствует ребро (v_i, v_j), 0 - иначе [6].

Тогда задача коммивояжера может быть сформулирована и решена в рамках линейного целочисленного программирования. Оптимальным маршрутом будем называть маршрут наименьшей длины:

$$f = \sum_{i \in I} \sum_{j \neq i,\, j \in I} c_{ij} x_{ij} \longrightarrow \min$$

С учетом условий

$$\sum_{j \neq i,\, j \in I} X_{ij} = 1, \forall i \in I \qquad (5)$$

$$\sum_{j \neq i,\, i \in I} X_{ij} = 1, \forall j \in I \qquad (6)$$

$$v_i - v_j + n x_{ij} \leq n-1 \ , \quad 1 \leq i \neq j \leq n \quad (7)$$

Ограничения (5) и (6) обеспечивают посещение узла маршрута однократно [7].

Неравенство (7) обеспечивает связность маршрута обхода пунктов, т. к. он не может состоять из двух и более не связных частей.

Анализ существующих подходов решения TSP

Начать анализ существующих алгоритмов решения задачи коммивояжера необходимо со следующего замечания: исходя из формулировки, очевидно, что эту проблему можно решить полным перебором всех доступных вариантов. Однако стоит заметить, что количество этих вариантов возрастает с ростом числа вершин I путевого графа и равно $I!$ [8]. Поэтому с ростом I будет многократно возрастать сложность вычислений, а значит алгоритмы, которые основаны на полном переборе, будут весьма неэффективны. При этом нужно отметить, что у методов полного перебора имеется существенное достоинство – это высокая по сравнению с другими алгоритмами точность полученного результата, что делает их вполне применимыми для решения небольших по вычислительному объему задач.

Но круг таких задач довольно узкий, поэтому обычно принимается решение отказываться от поисков точного решения задачи коммивояжера и предпринимаются попытки найти маршрут, максимально близкий к оптимальному. Для этой цели существует целая группа эвристических алгоритмов. Они находят решения, приближенные к оптимальному, но за гораздо меньшее по сравнению с точными методами время.

Количество городов	**Расчетное время**
10	1/3 секунды
13	8 минут
15	1 год
20	193 года

Табл.1 Расчетное время решения задачи TSP методом полного перебора

Особенностью многих эвристических алгоритмов является разнообразие решений, получаемых при применении алгоритма к одному и тому же примеру. Это обусловлено тем, что при построении маршрута различным вариантам продолжения маршрута соответствует вероятность их выбора, так как постоянный выбор наилучшего продолжения на каком-либо шаге алгоритма в итоге не всегда дает оптимальное решение. По причине больших расстояний, в рассматриваемых на практике задачах любое улучшение алгоритма позволит существенно уменьшить расхождение полученных решений с оптимальным.

Жадные алгоритмы

Жадный алгоритм (англ. Greedy algorithm) — алгоритм, заключающийся в принятии локально оптимальных решений на каждом этапе, при допущении, что конечное решение также окажется оптимальным. Самым известным алгоритмом этой группы являются алгоритм Хаффмана (адаптивный алгоритм оптимального префиксного кодирования алфавита с минимальной избыточностью) [9]. Также к жадным алгоритмам можно причислить алгоритм Дейкстры (простейший алгоритм по нахождению кратчайшего пути на связном графе).

Достоинством данной группы методов является время поиска решения, а недостатком – то, что решение часто является неоптимальным.

Отметим, что не существует возможности оценить применимость жадных алгоритмов в решении конкретной прикладной задачи, однако для задач, решаемых такими методами, выделяют две особенности:

1) К ним применим так называемый «Принцип жадного выбора»;
2) Они обладают свойством «Оптимальности для подзадач».

К задаче возможно применить принцип жадного выбора в том случае, когда последовательность локальных оптимумов является глобальным

оптимальным решением [10]. Доказательство оптимальности выглядит следующим образом:

1. Показываем, что жадность на первой итерации алгоритма не противоречит оптимальному решению;

2. Доказываем, что следующая итерация алгоритма идентична начальной;

3. Применяем принцип математической индукции.

Стоит обратить внимание на следующее обстоятельство: поскольку задача коммивояжера относится, к классу NP задач, то жадные алгоритмы не дают оптимального решения для них. Тем не менее в ряде случаев они дают очень неплохие приближенные решения, а с учетом довольно простой программной реализации и гораздо более меньших вычислительных затрат, эти методы находят свое применение при решении широкого круга задач транспортной логистики [11]. Блок-схема для жадного алгоритма изображена на рисунке 1.

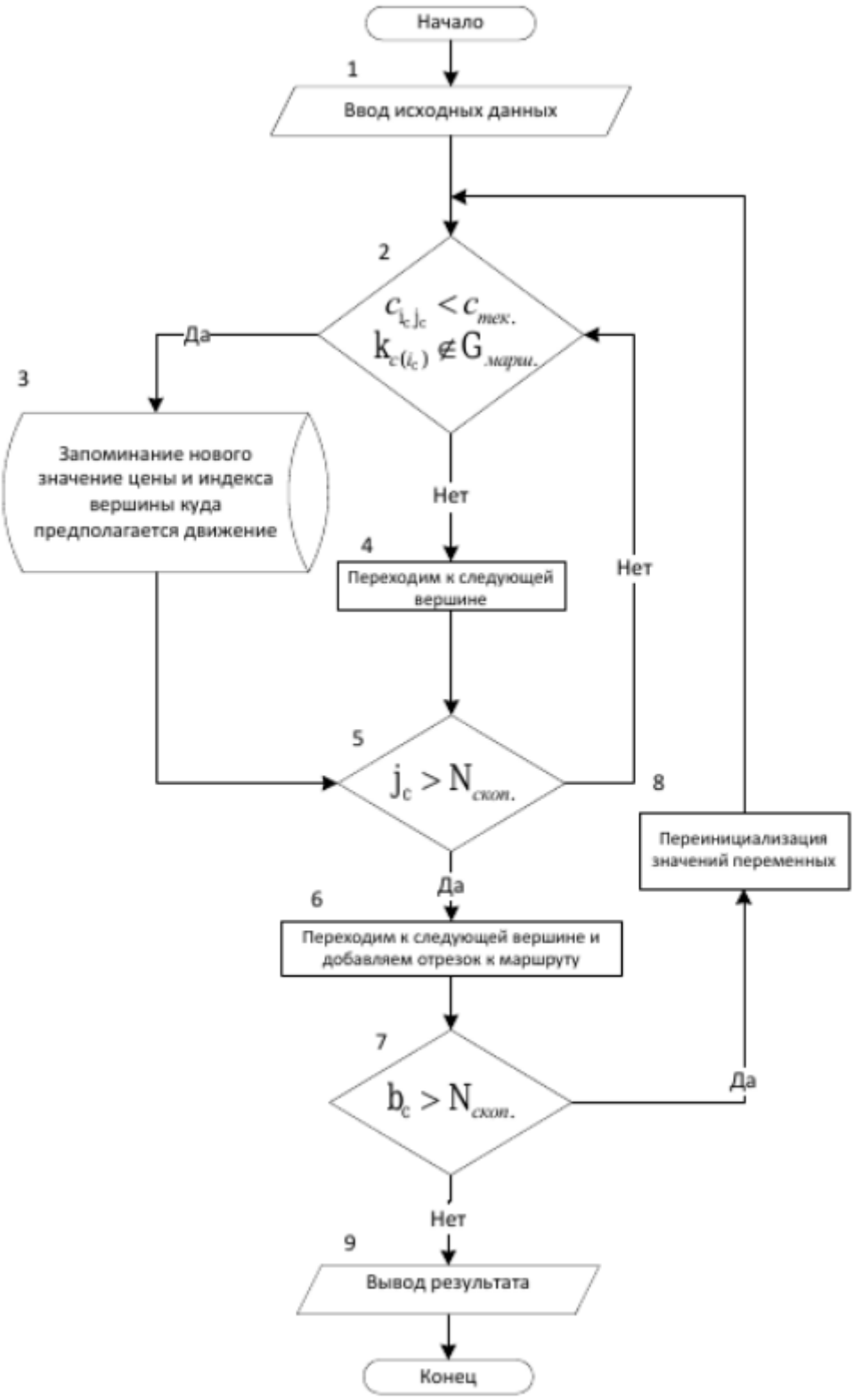


Рис.1 Блок-схема жадного алгоритма

Муравьиный алгоритм

Муравьиный алгоритм (алгоритм оптимизации подражанием муравьиной колонии, англ. ant colony optimization, ACO) - сравнительно молодой алгоритм. Первая версия алгоритма, предложенная доктором наук Марко Дориго в 1992 году, была направлена на поиск оптимального пути в графе [12]. Суть алгоритма заключается в анализе и использовании модели поведения муравьев, ищущих пути от колонии к источнику питания и представляет собой метаэвристическую оптимизацию.

В реальном мире муравьи двигаются хаотично и после нахождения источника пищи возвращаются в муравейник, помечая свой путь феромонами. Когда остальные муравьи находят такие феромоны, они идут по ним. Если в конце такого маршрута действительно находится источник питания, они укрепляют такой маршрут. Со временем феромоны испаряются, тем самым уменьшая привлекательность данного маршрута. Очевидно, что чем дольше преодолеваемый маршрут, тем меньшая концентрация феромонов на нем. На коротком пути, для сравнения, прохождение будет более быстрым и как следствие, плотность феромонов остается высокой. Если бы испарение не происходило, то первый путь был бы самым привлекательным. В этом случае, исследования пространственных решений были бы ограниченными. Таким образом, когда один муравей находит (например, короткий) путь от колонии до источника пищи, другие муравьи, скорее всего пойдут по этому пути, и положительные отзывы в конечном итоге приводят всех муравьев к одному, кратчайшему, пути.

Таким образом, в основу алгоритма заложение поведение муравьиной колонии. Перенося логику движения муравьев на задачу коммивояжера, имеем следующее описание метода:

1) Размещаем муравьев в вершинах заданного путевого графа (городах);

2) Начинаем движение муравьев.

В качестве направления движения этих муравьев используем вероятность, рассчитанную по формуле:

$$P_i = \frac{l_i^q * f_i^p}{\sum_{k=0}^{N} l_k^q * f_k^p}$$

где:

P_i – вероятность перехода по пути i,

l_i – величина, обратная весу i-го перехода,

f_i – количество феромона на i-м переходе,

q и p – эвристические параметры, обозначающие «жадность» и «стадность» алгоритма $(q + p = 1)$;

Решение, добытое таким алгоритмом, как и в прочих случаях, не будет точным. Но при удачном подборе параметров q и p можно добиться наиболее приближенного к оптимальному результата.

Очевидно, что одним из самых популярных применений муравьиного алгоритма является логистика транспортных перевозок, в частности с помощью этого метода успешно решается как раз разбираемая нами задача коммивояжера и задача о маршрутизации автотранспорта [22].

Метод ветвей и границ

Самым популярным алгоритмом в группе методов неявного перебора является метод ветвей и границ. В его основу лег принцип последовательного разбиения множества допустимых решений. На каждом шаге алгоритма элементы подмножества решений проходят проверку на наличие оптимума среди них. В случае решения задачи минимизации производится сравнение

нижней оценки значения целевой функции на данном подмножестве с верхней оценкой функционала. В качестве оценки сверху используется значение целевой функции на некотором допустимом решении.

Для более широкого понимания введем ряд определений. *Рекорд* – допустимое решение, дающее наименьшую верхнюю оценку. Если оценка снизу целевой функции на данном подмножестве не меньше оценки сверху, то рассматриваемое подмножество не содержит решения лучше рекорда и может быть отброшено. Если значение целевой функции на очередном решении меньше рекордного, то происходит смена рекорда. Будем говорить, что подмножество решений *просмотрено*, если установлено, что оно не содержит решения лучше рекорда. Если просмотрены все элементы разбиения, алгоритм завершает работу, а текущий рекорд является оптимальным решением. В противном случае среди непросмотренных элементов разбиения выбирается множество, являющееся в определенном смысле перспективным. Оно подвергается разбиению (ветвлению). Новые подмножества анализируются по описанной выше схеме. Процесс продолжается до тех пор, пока не будут просмотрены все элементы разбиения.

Общая идея алгоритма выглядит следующим образом:

Имеется задача:

$f(x) \to \min(x \in D)$, где $f(x)$ – вещественная функция, а D – конечное множество допустимых решений. Пусть $d \subseteq D$. Функцию $b(d)$, ставящую в соответствие множеству d разбиение его на подмножества $d_1,...,d_n, N > 1$, будем называть *ветвлением*. Вещественная функция $H(d)$ называется *нижней границей* для d, если

1) $H(d) \le f \min(x)$

2) на одноэлементном множестве $\{x\}$ верно равенство $H(\{x\}) = f(x)$

Алгоритм, реализующий метод ветвей и границ, состоит из последовательности однотипных шагов. На каждом шаге известен рекорд x^0 и подмножества $t_1, t_2,..., t_L$ непросмотренных решений. В начале работы алгоритма

$L=1$, $t_1=D$, x^0 – произвольный элемент множества D или пустое множество (на пустом множестве положим значение функционала равным бесконечности). На каждом шаге алгоритм начинает работу с проверки элементов разбиения. Пусть проверяется множество t_j. Множество t_j отсекается в одном из двух, последовательно проверяемых случаев:

а) если $H(t_j) \geq f(x^0)$;

б) если $H(t_j) < f(x^0)$ и найден такой элемент $y_j \in t_j$, что $f(y_j) = \min\ f(x) = H(t_j)$.

В случае «б» происходит смена рекорда $x^0 = y_j$. Пусть $t_1, t_2, ..., t_M$ $(M \leq L)$ – неотсеченные множества (будем считать, что отсечены множества с номерами $M+1, ..., L$). В случае $M=0$ алгоритм заканчивает работу, и в качестве решения задачи принимается рекорд x^0. При $M \geq 1$ среди множеств $t_1, t_2, ..., t_M$ выбирается множество для нового ветвления. Пусть таковым является множество t_1. Тогда осуществляется ветвление $b(t_1) = (d_1, ..., d_N)$, в результате которого получаем список множеств $d_1, ..., d_N$, $t_2, ..., t_M$. Эти множества нумеруются числами от 1 до L, и начинается новый шаг алгоритма. Нетрудно убедиться в том, что описанный алгоритм находит оптимальное решение за конечное число шагов. Описанная последовательность действий является общей схемой метода ветвей и границ для решения задач на минимум. При решении конкретной задачи следует указать способы построения нижней и верхней оценок, метод ветвления, а также правило выбора перспективного множества для разбиения.

Метод имитации отжига

На этом алгоритме мы остановимся более подробно, так как его изучение и способы его улучшения и являются предметом работы.

В основе этого алгоритма лежит реальный физический процесс

кристаллизации вещества, применяемый в металлургии. Отжиг – это процесс остывания вещества, при котором молекулы на фоне замедляющегося теплового движения собираются в наиболее энергетически выгодные конфигурации [14]. Как известно, у металла есть кристаллическая решетка, она описывает положение атомов в веществе. В этом случае маршрутом будет являться совокупность всех позиций атомов. Цель отжига – привести маршрут в состояние наименьшей протяженности. В самом начале алгоритма задается температура, с которой и начнется «отжиг», затем происходит постепенное ее понижение. Молекулы начинают выстраиваться в состояние с наименьшей протяженностью, но вследствие хаотичного теплового движения эта длина маршрута между ними может резко вырасти с определенной вероятностью. Для того чтобы уменьшить эту вероятность и понижается температура отжига. В конце концов, процесс будет завершен, когда температура упадет до заданного в алгоритме значения - Вещество остыло в точке с минимальной энергией.

Такая сложная схема с вероятностями переходов из точки в точку нужна потому, что, как было замечено раньше, эвристические алгоритмы имеют склонность к зацикливанию на локальном минимуме и выдаче его за глобальный оптимум. Чтобы выйти из такой ситуации, необходимо время от времени повышать энергию системы. При этом общая тенденция к поиску наименьшей энергии сохраняется. В этом и состоит суть метода имитации отжига [15].

Блок-схема этого алгоритма представлена на рисунке 2.

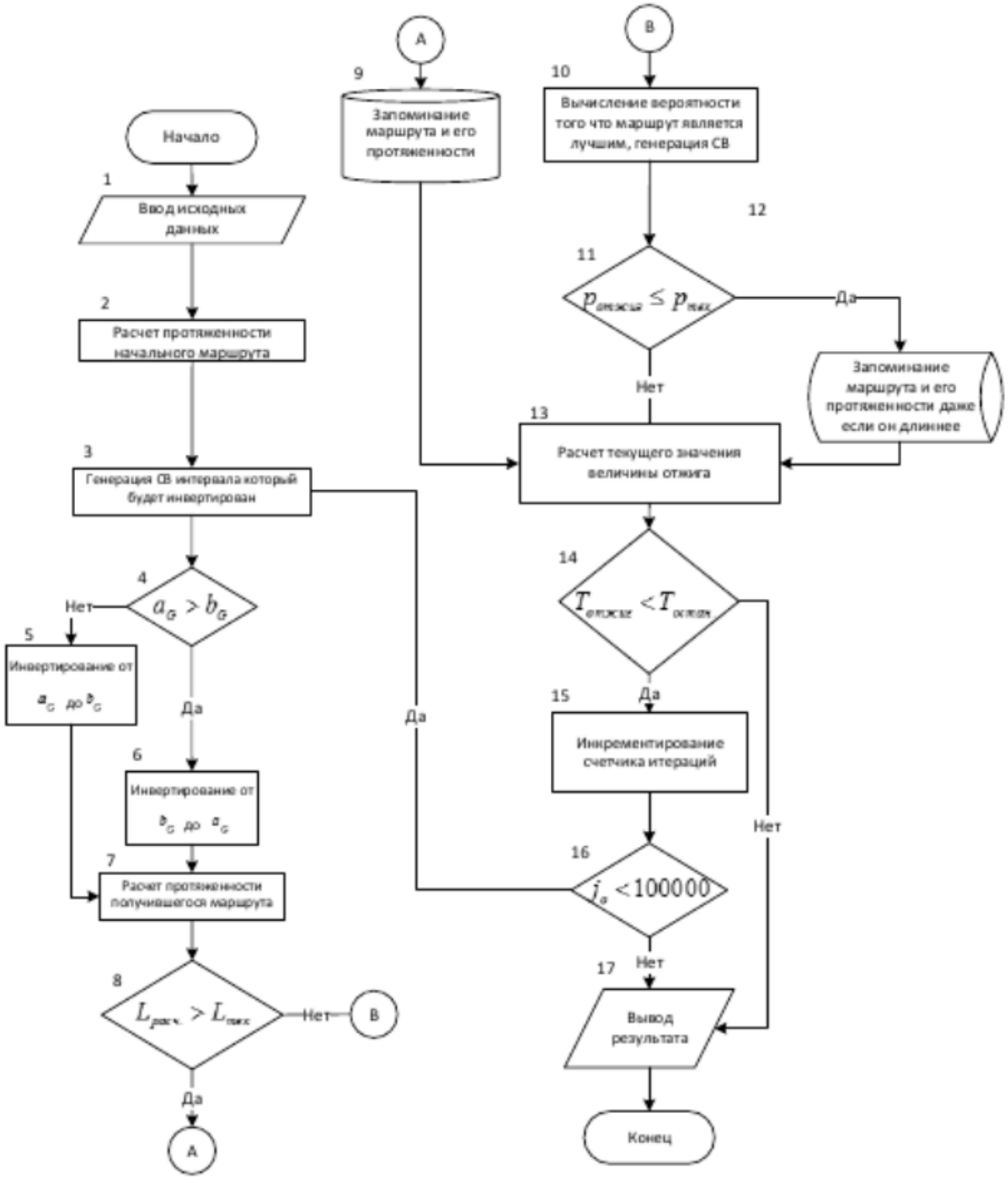


Рис.2 Блок-схема алгоритма метода имитации отжига

Математическое описание метода отжига

Чтобы описать этот алгоритм, нам понадобятся следующие обозначения:

S – множество всех состояний системы.

$f(S)$ – функция изменения состояния.

S_i – состояние системы на i - м шаге.

S_n – новое состояние (кандидат).

$T_{\min}$, $T_{\max}$ – минимальная и исходная температуры соответственно.

$f(T)$ – функция изменения температуры.

K – значение целевой функции.

Система начинает работать с исходного состояния S_1, начальной температуры $T_1 = T_{\max}$ и с заданной минимальной температурой $T_{\min}$. [15]

Пока $T_i > T_{\min}$ повторять действия:

1) $S_n = f(S_{i-1})$

2) $\Delta K = K(S_n) - K(S_{i-1})$

3) Если $\Delta K \le 0$, тогда $S_{i-1} = S_n$.

4) Иначе принятие нового состояния осуществляется с некоторой вероятностью $e^{(-\frac{\Delta}{T})}$.

5) Выбрать случайное число R на интервале $(0,1)$.

6) Если $e^{(-\frac{\Delta}{T})} > R$ осуществить переход $S_{i-1} = S_n$ иначе перейти к следующему шагу.

7) Снизить температуру T : $T_{i+1} = f(T_i)$.

8) Вернуть последнее состояние S.

Источником исходных данных послужил ресурс оф. сайт университета Waterloo [22], где уже были проведены вычисления оптимальных маршрутов через города различных стран. На основе этих данных были спроектированы города, через которых строились маршруты.

Город		Djibouti	Qatar	Luxembourg	Western Sahara	Uruguay
Количество вершин, шт		38	194	980	29	734
Наилучший маршрут, км		6656	9352	11340	27603	79114
Полный перебор	Длина, км	6656	N/A	N/A	27603	N/A
	Время, мин	86	N/A	N/A	45	N/A
Жадный алгоритм	Длина, км	6656	9352	N/A	27603	N/A
	Время, мин	82	379	N/A	33	N/A
Метод отжига	Длина, км	7112	10135	12798	28443	84235
	Время, мин	5	31	125	3	101
Метод ветвей и границ	Длина, км	7043	10266	13147	28388	85011
	Время, мин	8	34	140	4	122

Табл.2 Сравнение работы выбранных видов алгоритмов на пяти тестовых задачах из банка TSPLIB.

Динамическая адаптация метода отжига

Временная состоятельность

Результатом работы эвристических алгоритмов является приемлемое для большинства случаем решение задачи, но при этом его оптимальность не гарантируется. Это значит, что в данном случае происходит нарушение принципа оптимальности Беллмана, который утверждает, что любое дальнейшее решение, вне зависимости от первоначального состояния системы, должно продолжать быть оптимальным [19]. То есть на любых подзадачах оптимальное решение должно сохранять оптимальность - именно это свойство понимают под временной состоятельностью. С другой стороны, уменьшение размерности задачи повышает вероятность нахождения оптимального результата за меньшее время. Таким образом, лучшее решение все еще вычисляется с помощью эвристического алгоритма. Поскольку решение задачи коммивояжера - последовательность точек, посещенных транспортом, то к нему возможно применить оценку временной состоятельности [17].

Обозначим P - множество тестовых примеров для задачи, $p \in P$ - соответственно один пример из этого множества, $s(p)$ - множество решений, полученное при помощи эвристического алгоритма для задачи. Каждое такое решение является порядок обхода городов для каждого транспортного средства в течение T периодов, эта величина постоянна. Исходное решение $s(p)$ разделим на части, соответствующие каждому из периодов $t = 0,1,...,T-1$. Количество узлов, пройденных в течение первых t периодов, вычисляется по формуле $n(t, s(p)) = \frac{n_0 * t}{T}$, где n_0 – количество узлов в тестовой задаче, $s^+(t, p)$ - часть маршрута $s(p)$, соответствующая порядку обхода узлов после периода t, а $s^-(t, p)$ – до периода t [20]. Тогда каждое решение можно представить в виде объединения двух частей $s(p) = s^+(t, p) \cup s^-(t, p)$. Рассмотрим текущую задачу $\mathrm{p}(\mathrm{s}^-(t, p))$, которая отличается от начальной тестовой тем, что сокращено множество узлов, которые уже входят в часть маршрута $s^-(t, p)$. Через

$s(p(s^{-}(t,p)))$ обозначим решение текущей задачи, найденное при помощи эвристического алгоритма. Назовем решение $s(p)$ *состоятельным по времени*, если для каждого $t = 0,1,...,T-1$ верно следующее неравенство:

$f(s^{+}(t,p)) \le fs(p(s^{-}(t,p)))$, где f – целевая функция рассматриваемой задачи [18].

Для каждого решения $s(p)$ нужно провести M вычислительных экспериментов, цель которых состоит в проверке решения на состоятельность по времени. Через $b(s(p),t)$ обозначим число экспериментов, для которых нарушается это свойство после периода t. Если решение является оптимальным, то, по критерию оптимальности Беллмана, выполняется равенство $\sum_{k=1}^{T-1} b(s(p),t) = 0$.[19]

Экспериментальным уровнем временной состоятельности алгоритма называется величина, определяемая по следующей формуле

$$con\ (S(p)) = 1 - \frac{1}{M|P|}\sum_{p\in P}\frac{1}{S(p)}\sum_{k=1}^{T-1} b(s(p),t),$$

где $S(p)$ – множество решений, полученных в результате обработки эвристикой задачи p [20]. Очевидно, что $con \in [0;1]$. В зависимости от значения *con* можно делать вывод о соблюдении принципа Беллмана: чем ближе оно к 1, тем больше вероятность того, что решение сохранит свойство оптимальности.

Описание работы динамической адаптации

Метод динамической адаптации алгоритмов для задачи коммивояжера подробно был рассмотрен в статье [20]. Используя идеи из этой статьи, применим его для нашего случая. Динамическую адаптацию можно описать следующим образом: при старте метода для всех узлов задачи должно быть сгенерировано N решений, после чего выбирается решение, у которого целевая функция имеет минимальное значение. В дальнейшем каждый маршрут делится на T периодов, после каждого из которых происходит уменьшение количества узлов. При смене периода алгоритм повторяется. Маршрут требуется изменить, есть на очередной итерации алгоритма будет найдено решение, лучшее текущего. Прогресс в реализации метода достигается за счёт того, что на каждом шаге количество рассматриваемых узлов сокращается. [21] Общий алгоритм метода имитации отжига при применении динамической адаптации выглядит следующим образом:

1. Формируем N результатов выполнения алгоритма;
2. Определяем лучший результат из найденных;
3. От $t = 1$ до $t < T$ НАЧАЛО ЦИКЛА
    4. Формируем следующие n результатов;
    5. Выбираем результат с наименьшей целевой функцией $f(p')$;
    6. ЕСЛИ $f(p') \le f(p)$ тогда принимаем результат p', ИНАЧЕ продолжаем выполнение алгоритма;
7. КОНЕЦ ЦИКЛА

Применение динамической адаптации

В вычислительном эксперименте используем значение параметра $T = 3$. За множество задач P возьмем задачу TSP с различным количеством узлов, т. е. $|P| = 5$. Для каждой задачи, сгенерировано по 10 решений, количество тестов M для одного решения $s(p)$ равно 5.

Из этих данных рассчитаем средний экспериментальный уровень временной состоятельности для алгоритма имитации отжига $con = 0.314$.

Исходя из полученного значения, делаем вывод о том, что больше трети начальных решений удовлетворяют критерию оптимальности в процессе своей реализации, а значит, могут существовать и другие маршруты с меньшим значением целевой функции, в сравнении с начальным решением. Реализацию динамической адаптации алгоритма на языке Javascript можно найти в Приложении.

Ниже представлены результаты динамической адаптации алгоритма имитации отжига для задач с различным количеством узлов при $T = 3$

Количество тестов	Значения временных периодов			Количество состоятельных по времени маршрутов
100	1	2	3	13
	48	29	10	

Табл.3 Оценка уровня временной состоятельности для базового алгоритма имитации отжига при решении задачи TSP

Количество тестов	Значения временных периодов			Количество состоятельных по времени маршрутов
100	1	2	3	49
	17	25	9	

Табл.4 Оценка уровня временной состоятельности для динамически адаптированного алгоритма имитации отжига при решении задачи TSP

При анализе получившихся результатов отметим, что в подавляющем большинстве случаев адаптированный алгоритм показывает лучший результат. В случаях с небольшим начальным массивов городов, разница не так заметна, так как решение, добытое неадаптированных алгоритмом, уже максимально близко к оптимальному, однако при росте размерности входных данных мы наблюдаем растущую эффективность, так как при увеличении размерности эффективность алгоритма имитации отжига падает, и возникает возможность оптимизации алгоритма. Коэффициент оптимизации решения при применении динамической адаптации в сравнении с первоначальным алгоритмом находится в пределах $3-15.6\%$.

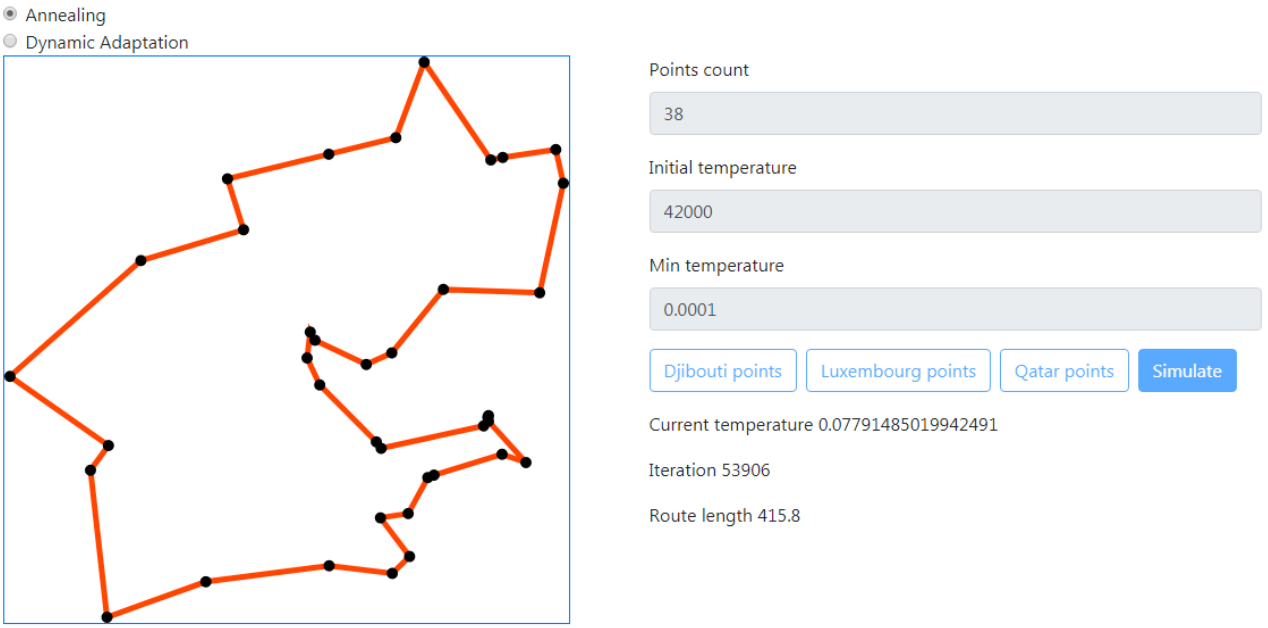


Рис.3 Результат работы алгоритма на примере данных из банка TSPLIB dj38 (Djibouti). Координаты неотмасштабированы

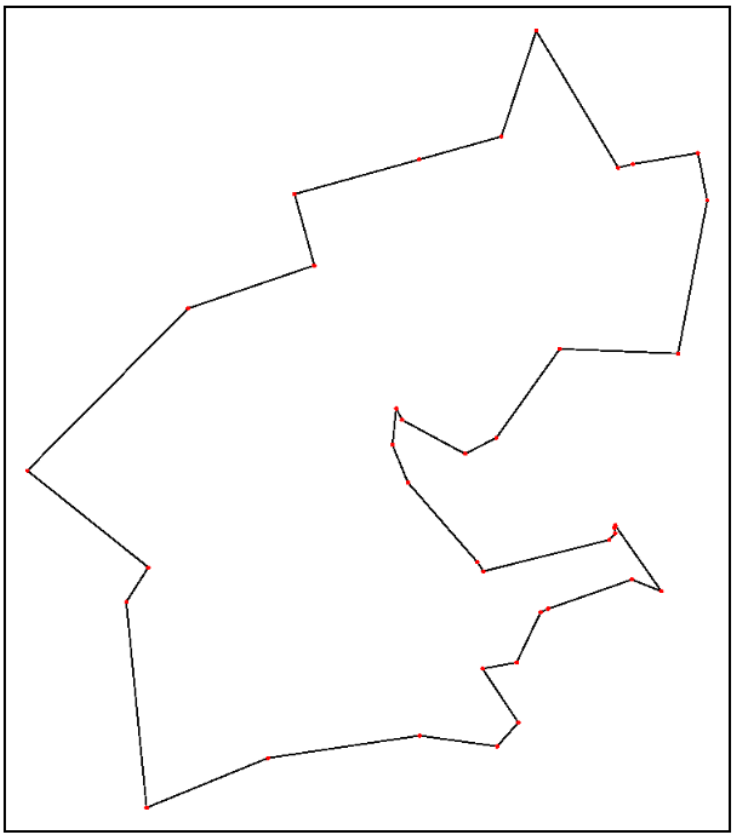


Рис.4 Оптимальный маршрут. Источник: оф. Сайт университета Waterloo [22]. TSPLIB dj38 (Djibouti)

Город		Djibouti	Qatar	Luxembourg	Western Sahara	Uruguay
Количество вершин, шт		38	194	980	29	734
Наилучший маршрут, км		6656	9352	11340	27603	79114
Метод имитации отжига	Длина, км	7112	10135	12798	28443	84235
	Время, мин	5	31	125	3	101
Динамическая адаптация	Длина, км	6857	9633	11680	28012	82177
	Время, мин	4	25	88	2	71

Табл.5 Результаты работы программы

Заключение

В данной работе были исследованы различные эвристические подходы, необходимые для решения задач данного типа. Среди изученных подходов выбор был остановлен на методе имитации отжига. Реализация метода была осуществлена на языке Javascript и рассмотрена на тестовых данных. Проанализировав полученные данные, было выявлено преимущество использования эвристики перед жадным алгоритмом и полным перебором. К тому же, эвристика не дает оптимального решения, что было показано при вычислении оценки уровня временной состоятельности алгоритма, среднее значение величины которого равно 0.314. В ходе проведения работы был разработана процедура динамической адаптации метода имитации отжига, впоследствии реализованная в программном коде. При проверке полученных результатов было выявлено, что значение решения, сгенерированного данным алгоритмом, меньше, чем алгоритмом имитации отжига. Улучшение значения решения в экспериментах для задачи с различным количеством узлов принимает до 15,6%. Результаты показали, что использование на практике алгоритма динамической адаптации позволяет генерировать маршруты меньшей длины, и, соответственно, меньшие по времени.

Список используемой литературы

1. Введение в оптимизацию. Имитация отжига. https://habrahabr.ru/post/209610/

2. Борознов Владимир Олегович. Исследование решения задачи коммивояжера. Вестник Астраханского государственного технического университета. Серия: Управление, вычислительная техника и информатика. Выпуск № 2 / 2009

3. Гараба И.В. Сравнительный анализ методов решения задачи коммивояжера для выбора маршрута прокладки кабеля сети кольцевой архитектуры. Молодежный научно-технический вестник. УДК 519.173

4. Задача коммивояжера и методы решения. http://lmatrix.ru/news/practice/zadacha-kommivoyazhera-i-metody-resheniya-vse-o-logistike_36.html

5. A simulated annealing approach to explore temporal consolidation of healthcare courier services to reduce carbon emissions. INSPEC Accession Number**:** 14761720. Published in: Service Operations and Logistics, and Informatics (SOLI), 2014 IEEE International Conference.

6. Simulation-based optimization using simulated annealing for optimal equipment selection within print production environments. INSPEC Accession Number: 14060090. Published in: Simulation Conference (WSC), 2013 Winter

7. Vasek Chvatal, William J. Cook, George B. Dantzig, Delbert Ray Fulkerson,and Selmer M. Johnson. Solution of a large-scale traveling-salesman problem. In 50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art, pages 7–28. 2010.

8. Майника Э. Алгоритмы оптимизации на сетях и графах. М: Мир, 1981. 323 с.

9. Christos H. Papadimitriou and Kenneth Steiglitz. Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.

10. Носкова Екатерина. Метод имитации отжига для задачи составления расписаний с параллельными процессорами. Научный корреспондент, 2016 г.

11. О. Оре Графы и их применение. Пер. с англ. под ред. И.М. Яглома. - М., “Мир”, 1965, 174 с.

12. В. П. Сигорский. Математический аппарат инженера. - К., “Техніка”, 1975, 768 с.

13. А.А.Ежов, С.А.Шумский. Нейрокомпьютинг и его применения в экономике и бизнесе

14. Ю. Н. Кузнецов, В. И. Кузубов, А. Б. Волощенко. Математическое программирование: учебное пособие. 2-е изд. перераб. и доп. - М.; Высшая школа, 1980, 300 с., ил.

15. Е. В. Маркова, А. Н. Лисенков. Комбинаторные планы в задачах многофакторного эксперимента. – М., Наука, 1979, 345 с.

16. Tannenbaum P. Excursions in Mathematics. University of Kansas, 2011. P. 25

17. В. М. Бондарев, В. И. Рублинецкий, Е. Г. Качко. Основы программирования. – Харьков, Фолио; Ростов на Дону, Феникс, 1998, 368 с.

18. Ф. А. Новиков Дискретная математика для программистов. - Санкт-Петербург, Питер, 2001, 304 с., ил.

19. Рейндогльд Э., Део Н. Комбинаторные алгоритмы решения задачи коммивояжера. Теория и практика – М.: Мир, 2000, 400с.

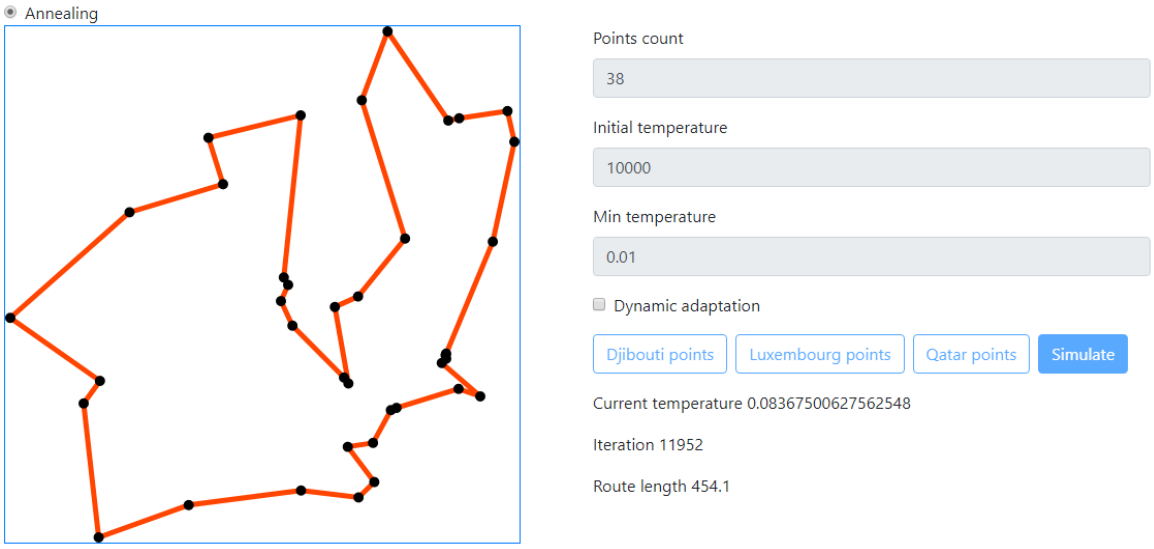
20. Shirokikh, V. A., Zakharov, V. V. Dynamic Adaptive Large NeighborhoodSearch for Inventory Routing Problem. // Advances in Intelligent Systemsand Computing, Vol. 359, 2015.

21. Егорова А. А., Касимова Я. А., Арасланова В. А. Динамическая адаптация эвристического алгоритма для задачи транспортной маршрутизации при использовании кросс-докинга // Молодой ученый. — 2016. — №14. — С. 11-15.

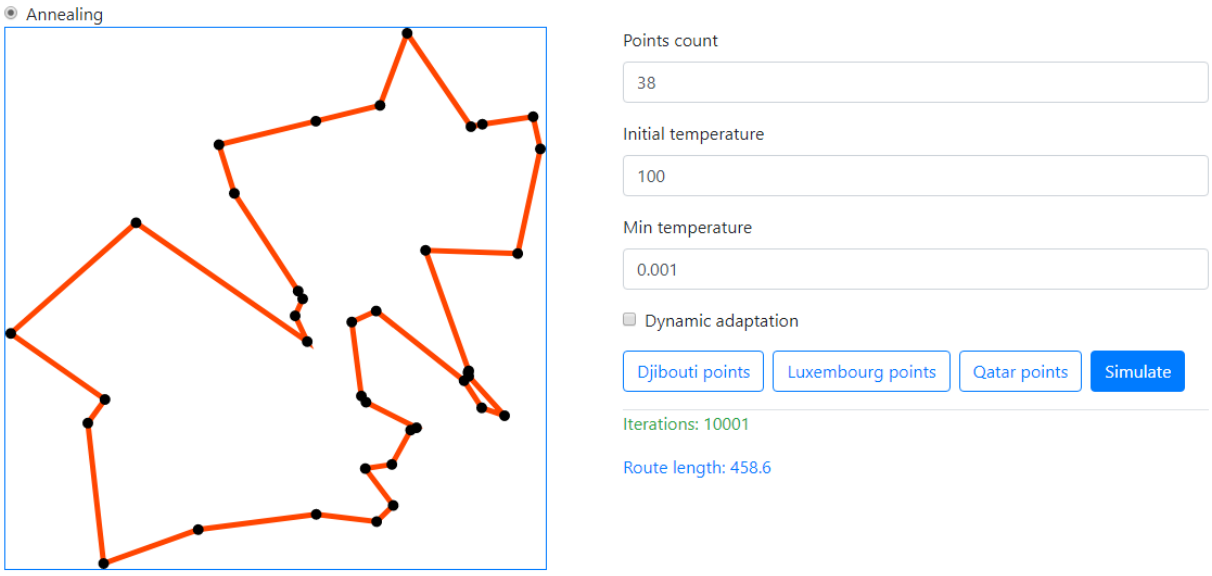
22. Natural Sciences and Engineering Research Council of Canada (NSERC) and the Department of Combinatorics and Optimization at the University of Waterloo. http://www.math.uwaterloo.ca/tsp/world/countries.html

23. Мугайских, А. В. (2015). Динамическая адаптация генетического алгоритма для задачи коммивояжёра. Процессы управления и устойчивость (Том 2, стр. 665-670)

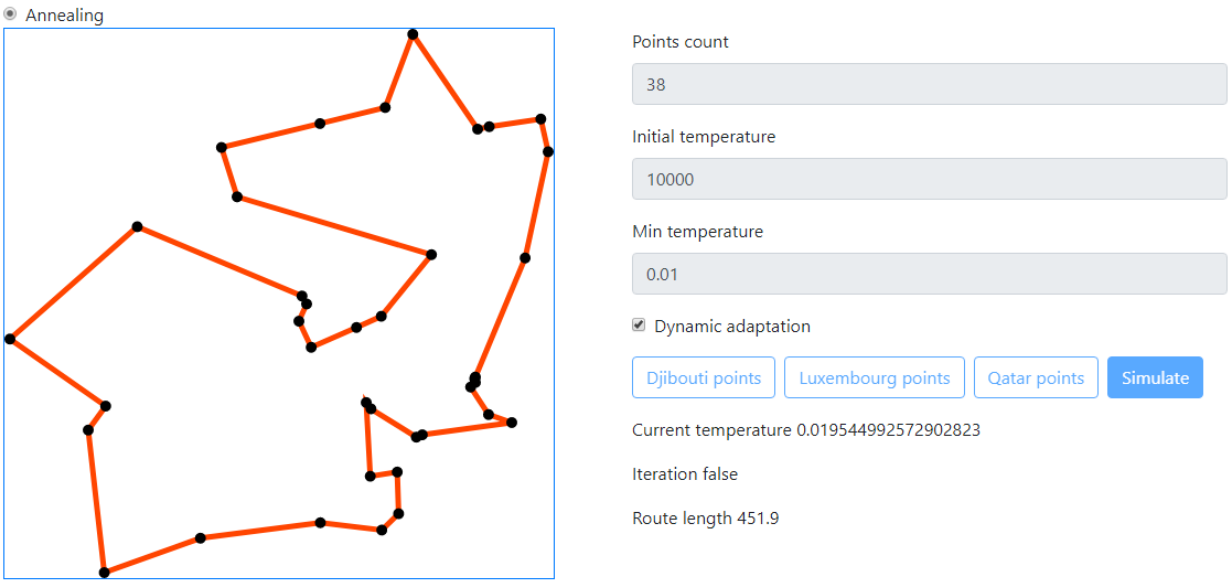
Приложение



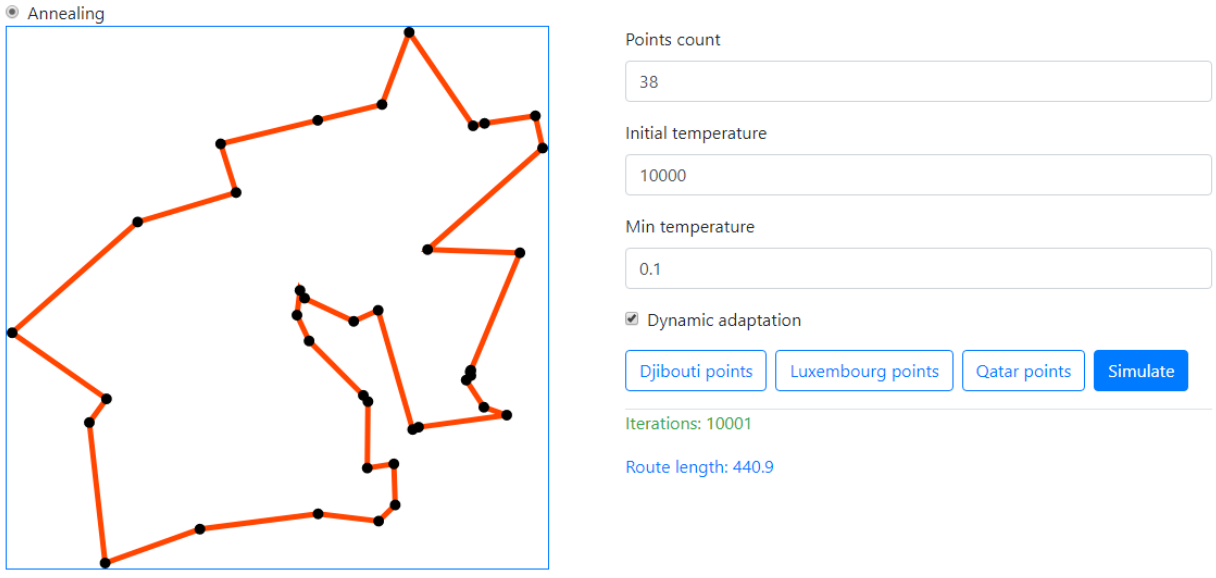
Построенный маршрут для Djibouti с использованием метода имитации отжига



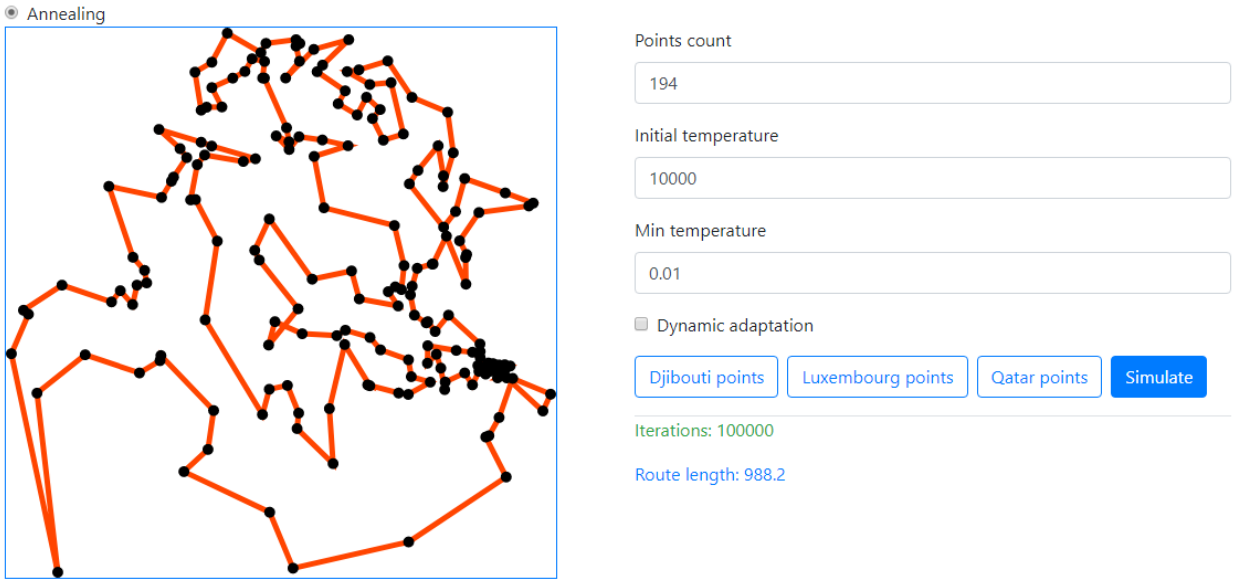
Построенный маршрут для Djibouti с использованием метода имитации отжига



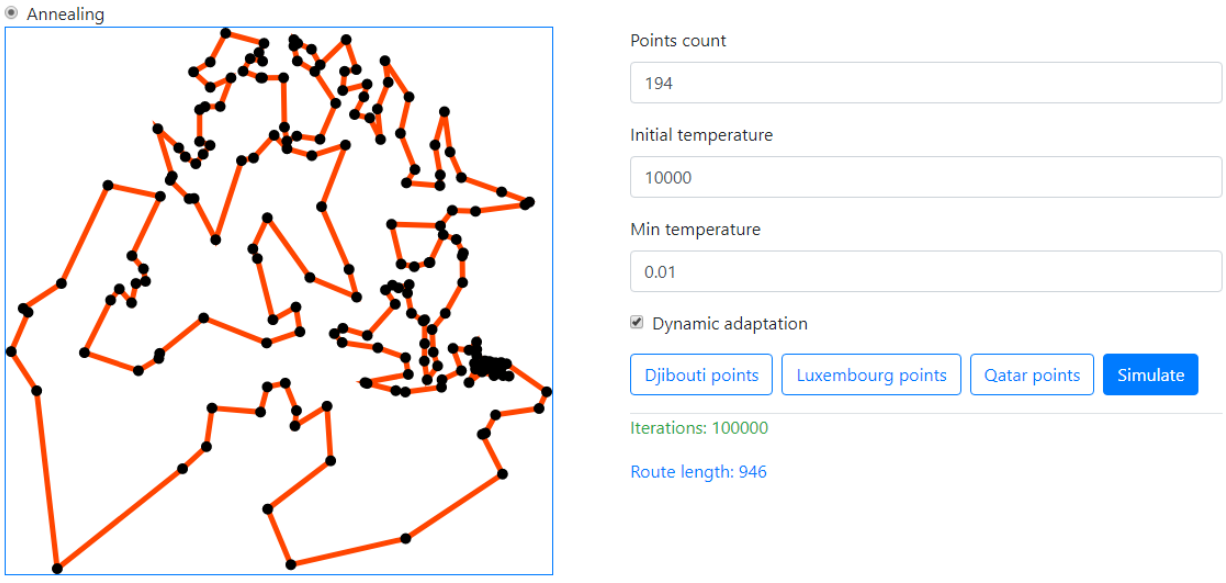
Построенный маршрут для Djibouti с использованием динамической адаптации метода имитации отжига



Построенный маршрут для Djibouti с использованием динамической адаптации метода имитации отжига



Построенный маршрут для Qatar с использованием метода имитации отжига



Построенный маршрут для Qatar с использованием динамической адаптации метода имитации отжига

Исходный код программы:

Annealing.js

```
Vue.component('annealing', {
  name: 'Annealing',
  template:
    '<div class="row">' +
      '<div class="col-6">' +
          '<points :points="points" :route="route" class="border border-primary mb-3" style="width:
500px;height: 500px;"></points>' +
      '</div>' +
      '<div class="col-6">' +
          '<div class="form-group">' +
              '<label for="it">Points count</label>' +
              '<input :disabled="simulated" type="text" class="form-control" id="p" v-
model="pointsCount" placeholder="Enter points count">' +
          '</div>' +
          '<div class="form-group">' +
              '<label for="it">Initial temperature</label>' +
              '<input :disabled="simulated" type="text" class="form-control" id="it" v-
model="initialTemperature" placeholder="Enter initial temperature">' +
          '</div>' +
          '<div class="form-group">' +
              '<label for="mt">Min temperature</label>' +
              '<input :disabled="simulated" type="text" class="form-control" id="mt" v-
model="minTemperature" placeholder="Enter min temperature">' +
          '</div>' +

          '<div class="form-check mb-3">' +
            '<input class="form-check-input"' +
              'type="checkbox" value="" id="dynamicAdaptation" v-model="dynamicAdaptation">' +
            '<label class="form-check-label" for="dynamicAdaptation">' +
              'Dynamic adaptation' +
            '</label>' +
          '</div>' +
          // '<div v-if="dynamicAdaptation" class="form-group">' +
          //     '<label for="pc">Periods count</label>' +
          //     '<input :disabled="simulated" type="text" class="form-control" id="pc" v-
model="periodsCount" placeholder="Enter periods count">' +
          // '</div>' +
          // '<div v-if="dynamicAdaptation" class="form-group">' +
          //     '<label for="sc">Solutions count</label>' +
          //     '<input :disabled="simulated" type="text" class="form-control" id="sc" v-
model="solutionsCount" placeholder="Enter solutions count">' +
          // '</div>' +

          '<button :disabled="simulated" class="btn btn-outline-primary mr-2"
@click="handleDjiboutiPoints">Djibouti points</button>' +
          '<button :disabled="simulated" class="btn btn-outline-primary mr-2"
@click="handleLuxembourgPoints">Luxembourg points</button>' +
          '<button :disabled="simulated" class="btn btn-outline-primary mr-2"
@click="handleQatarPoints">Qatar points</button>' +
          // '<button :disabled="simulated" class="btn btn-primary mr-2"
@click="handlePoints">Points</button>' +
          // '<button :disabled="simulated" class="btn btn-primary mr-2"
@click="handleRoute">Route</button>' +
          '<button :disabled="simulated" class="btn btn-primary"
@click="handleSimulate">Simulate</button>' +
          '<div v-if="simulated" class="mt-3">' +
              '<p>Current temperature {{currentTemperature}}</p>' +
              '<p>Iteration {{iteration}}</p>' +
              '<p>Route length {{energy}}</p>' +
          '</div>' +
          '<div v-if="iteration && !simulated" class="border-top mt-3 text-success">' +
              '<p>Iterations: {{iteration}}</p>' +
          '</div>' +
          '<div v-if="energy && !simulated" class="text-primary">' +
              '<p>Route length: {{energy}}</p>' +
          '</div>' +
      '</div>' +
    '</div>',
  data () {
    return {
      points: [],
      route: [],
      worker: new Annealing(),
      simulated: false,
```

```
    pointsCount: 100,
    initialTemperature: 10,
    minTemperature: 0.001,
    currentTemperature: NaN,
    iteration: false,
    energy: false,
    dynamicAdaptation: false,
    periodsCount: 5,
    solutionsCount: 10
  }
},
watch: {
  // pointsCount: function () {
  //   if (this.points.length > 0) {
  //     this.generatePoints()
  //     this.route = this.worker.generateRoute(this.points.length)
  //   }
  // }
  dynamicAdaptation: function (val) {
    if (val) {
      this.worker = new DynamicAdaptation(new Annealing())
    } else {
      this.worker = new Annealing()
    }

    this.initWorker()
  }
},
mounted () {
  this.initWorker()
},
methods: {
  // handlePoints () {
  //   this.generatePoints()
  // },
  // handleRoute () {
  //   this.route = this.worker.generateRoute(this.points.length)
  // },
  initWorker () {
    this.worker.on('updateRoute', route => {
      this.route = route
    })

    this.worker.on('updateTemperature', t => {
      this.currentTemperature = t
    })

    this.worker.on('newIteration', i => {
      this.iteration = i
    })

    this.worker.on('updateEnergy', i => {
      this.energy = Math.round(i * 10) / 10
    })
  },
  getPoints (dataSet) {
    const x1 = dataSet.xMin
    const xn = dataSet.xMax
    const y1 = dataSet.yMin
    const yn = dataSet.yMax

    // console.log(Math.min(...qatar.points.map(({ y }) => y)))
    // console.log(Math.max(...qatar.points.map(({ y }) => y)))
    //
    // console.log(Math.min(...qatar.points.map(({ x }) => x)))
    // console.log(Math.max(...qatar.points.map(({ x }) => x)))

    return dataSet.points.map(({ x, y }) => {
      return new Point((x - x1) * (98 / (xn - x1)) + 1, 100 - ((y - y1) * (98 / (yn - y1)) + 1))
    })
  },
  handleDjiboutiPoints () {
    this.points = this.getPoints(djibouti)

    this.pointsCount = this.points.length
  },
  handleLuxembourgPoints () {
    this.points = this.getPoints(luxembourg)
```

```
      this.pointsCount = this.points.length
    },
    handleQatarPoints () {
      this.points = this.getPoints(qatar)

      this.pointsCount = this.points.length
    },
    async handleSimulate () {
      this.simulated = true

      if (this.points.length === 0 || this.points.length !== parseInt(this.pointsCount)) {
        this.generatePoints()
      }

      // this.points = [
      //   [98, 41],
      //   [16, 69],
      //   [27, 22],
      //   [3, 59],
      //   [26, 38],
      //   [12, 42],
      //   [32, 30]
      // ].map(i => new Point(i[0], i[1]))

      if (this.dynamicAdaptation) {
        this.worker.k = false
        await this.worker.simulate(
          this.points,
          this.periodsCount,
          this.solutionsCount,
          { initialTemperature: this.initialTemperature, minTemperature: this.minTemperature }
        )
      } else {
        this.worker.k = true
        await this.worker.simulate(this.points, this.initialTemperature, this.minTemperature)
      }

      this.simulated = false
    },
    generatePoints () {
      this.points = this.worker.getRandPoints({ size: this.pointsCount })
    }
  }
})
```

Dispatcher.js

```
class Dispatcher {
 constructor () {
  this.events = {}
 }

 dispatch (eventName, data) {
  const event = this.events[eventName]
  if (event) {
   event.fire(data)
  }
 }

 on (eventName, callback) {
  let event = this.events[eventName]
  if (!event) {
   event = new DispatcherEvent(eventName)
   this.events[eventName] = event
  }
  event.registerCallback(callback)
 }
```

```
  off (eventName, callback) {
    const event = this.events[eventName]
    if (event && event.callbacks.includes(callback)) {
      event.unregisterCallback(callback)
      if (event.callbacks.length === 0) {
        delete this.events[eventName]
      }
    }
  }
}
```

DispatcherEvent.js

```
class DispatcherEvent {
  constructor (eventName) {
    this.eventName = eventName
    this.callbacks = []
  }

  registerCallback (callback) {
    this.callbacks.push(callback)
  }

  unregisterCallback (callback) {
    this.callbacks = this.callbacks.filter(c => c !== callback)
  }

  fire (data) {
    const callbacks = this.callbacks.slice(0)
    callbacks.forEach(callback => {
      callback(data)
    })
  }
}
```

Djibouti.js (Luxembourg.js, Qatar.js – аналогично)

```
const djibouti = {
  yMin: 11003.611100,
  yMax: 12645.000000,
  xMin: 41836.1111,
  xMax: 43355.5556,
  points: [
    { x: 42102.500000, y: 11003.611100 },
    { x: 42373.888900, y: 11108.611100 },
    { x: 42885.833300, y: 11133.333300 },
    { x: 42712.500000, y: 11155.833300 },
    { x: 42933.333300, y: 11183.333300 },
    { x: 42853.333300, y: 11297.500000 },
    { x: 42929.444400, y: 11310.277800 },
```

```
  { x: 42983.333300, y: 11416.666700 },
  { x: 43000.277800, y: 11423.888900 },
  { x: 42057.222200, y: 11438.333300 },
  { x: 43252.777800, y: 11461.111100 },
  { x: 43187.222200, y: 11485.555600 },
  { x: 42855.277800, y: 11503.055600 },
  { x: 42106.388900, y: 11511.388900 },
  { x: 42841.944400, y: 11522.222200 },
  { x: 43136.666700, y: 11569.444400 },
  { x: 43150.000000, y: 11583.333300 },
  { x: 43148.055600, y: 11595.000000 },
  { x: 43150.000000, y: 11600.000000 },
  { x: 42686.666700, y: 11690.555600 },
  { x: 41836.111100, y: 11715.833300 },
  { x: 42814.444400, y: 11751.111100 },
  { x: 42651.944400, y: 11770.277800 },
  { x: 42884.444400, y: 11785.277800 },
  { x: 42673.611100, y: 11822.777800 },
  { x: 42660.555600, y: 11846.944400 },
  { x: 43290.555600, y: 11963.055600 },
  { x: 43026.111100, y: 11973.055600 },
  { x: 42195.555600, y: 12058.333300 },
  { x: 42477.500000, y: 12149.444400 },
  { x: 43355.555600, y: 12286.944400 },
  { x: 42433.333300, y: 12300.000000 },
  { x: 43156.388900, y: 12355.833300 },
  { x: 43189.166700, y: 12363.333300 },
  { x: 42711.388900, y: 12372.777800 },
  { x: 43334.722200, y: 12386.666700 },
  { x: 42895.555600, y: 12421.666700 },
  { x: 42973.333300, y: 12645.000000 }
 ]
```

DynamicAdaptation.js

```
Vue.component('dynamic-adaptation', {
 name: 'DynamicAdaptation',
 template:
  '<div class="row">' +
   '<div class="col-6">' +
     '<points :points="points" :route="route" class="border border-primary mb-3" style="width: 500px;height: 500px;"></points>' +
   '</div>' +
   '<div class="col-6">' +
     '<div class="form-group">' +
       '<label for="it">Points count</label>' +
       '<input :disabled="simulated" type="text" class="form-control" id="p" v-model="pointsCount" placeholder="Enter points count">' +
     '</div>' +
     '<div class="form-group">' +
       '<label for="pc">Periods count</label>' +
       '<input :disabled="simulated" type="text" class="form-control" id="pc" v-model="periodsCount" placeholder="Enter periods count">' +
     '</div>' +
     '<div class="form-group">' +
```

```
          '<label for="sc">Solutions count</label>' +
          '<input :disabled="simulated" type="text" class="form-control" id="sc" v-model="solutionsCount" placeholder="Enter solutions count">' +
        '</div>' +
        '<button :disabled="simulated" class="btn btn-primary mr-2" @click="handlePoints">Points</button>' +
        '<button :disabled="simulated" class="btn btn-primary" @click="handleSimulate">Simulate</button>' +
        '<div v-if="simulated" class="mt-3">' +
          // '<p>Current period {{currentPeriod}}</p>' +
          '<p>Iteration {{iteration}}</p>' +
          '<p>Route length {{energy}}</p>' +
        '</div>' +
        '<div v-if="iteration && !simulated" class="border-top mt-3 text-success">' +
          '<p>Iterations: {{iteration}}</p>' +
        '</div>' +
        '<div v-if="energy && !simulated" class="text-primary">' +
          '<p>Route length: {{energy}}</p>' +
        '</div>' +
    '</div>' +
  '</div>',
data () {
  return {
    points: [],
    route: [],
    worker: new DynamicAdaptation(new Annealing()),
    simulated: false,
    pointsCount: 100,
    periodsCount: 5,
    solutionsCount: 10,
    // currentPeriod: NaN,
    iteration: false,
    energy: false
  }
},
watch: {
  pointsCount: function () {
    if (this.points.length > 0) {
      this.generatePoints()
      // this.route = this.worker.generateRoute(this.points.length)
    }
  }
},
mounted () {
  this.worker.on('updateRoute', route => {
    this.route = route
  })

  // this.worker.on('updatePeriod', t => {
  //   this.currentPeriod = t
  // })

  this.worker.on('newIteration', i => {
    this.iteration = i
  })
```

```
    this.worker.on('updateEnergy', i => {
      this.energy = Math.round(i * 10) / 10
    })
  },
  methods: {
    handlePoints () {
      this.generatePoints()
    },
    async handleSimulate () {
      this.simulated = true

      if (this.points.length === 0) {
        this.generatePoints()
      }

      await this.worker.simulate(this.points, this.periodsCount, this.solutionsCount)

      this.simulated = false
    },
    generatePoints () {
      this.points = this.worker.getRandPoints({ size: this.pointsCount })
    }
  }
})
```

graph.js

```
Vue.component('points', {
  name: 'Points',
  template:
    '<div>' +
      '<svg viewBox="0 0 100 100" class="w-100 h-100">' +
        '<polyline fill="none" stroke="orangered" stroke-width="1" :points="lines"></polyline>' +
        '<circle v-for="p in points" r="1" :cx="`${p.x}%`" :cy="`${p.y}%`"></circle>' +
      '</svg>' +
    '</div>',
  props: {
    points: {
      type: Array,
      default: () => []
    },
    route: {
      type: Array,
      default: () => []
    }
  },
  computed: {
    lines: function () {
      if (this.points.length === 0 || this.route.length === 0) return ''

      const lines = this.route.map(i => `${this.points[i].x} ${this.points[i].y}`)
```

```
      lines.push(`${this.points[this.route[0]].x} ${this.points[this.route[0]].y}`)

      return lines.join(',')
    }
  }
})
```

lib.js

```
class Point {
  constructor (x, y) {
    this._x = x
    this._y = y
  }

  get x () {
    return this._x
  }

  get y () {
    return this._y
  }

  clone () {
    return new Point(this._x, this._y)
  }
}

const delay = milliseconds => {
  return new Promise(resolve => {
    setTimeout(resolve, milliseconds)
  })
}

class Annealing extends Dispatcher {
  constructor () {
    super()
    this._k = false
  }

  set k (val) {
    this._k = val
  }

  decreaseTemperature (initial, i) {
    return (initial * 0.1) / i
  }

  getTransitionProbability (dE, t) {
    return Math.exp(-(Math.abs(dE) / t))
  }
```

```
isTransition (p) {
  return Math.random() <= p
}

getRandomInt (min = 0, max = 1) {
  return Math.floor(Math.random() * (max - min)) + min
}

getRandomInclusive (min = 0, max = 1) {
  return Math.floor(Math.random() * (max - min + 1)) + min
}

metric (a, b) {
  return Math.sqrt((a.x - b.x) ** 2 + (a.y - b.y) ** 2)
}

getRandPoints (params) {
  const p = Object.assign({
    size: 100
  }, params)

  const points = []

  for (let i = 0; i < p.size; i++) {
    points.push(new Point(this.getRandomInt(1, 100), this.getRandomInt(1, 100)))
  }

  return points
}

generateStateCandidate (route) {
  if (this._k && this.getRandomInt(0, 10) > 5) {
    return route
  }

  const n = route.length

  let i = this.getRandomInt(0, n)
  let j = this.getRandomInt(0, n)

  // if (i === j) return this.generateStateCandidate(route)

  if (i > j) [i, j] = [j, i]

  return [...route.slice(0, i), ...route.slice(i, j + 1).reverse(), ...route.slice(j + 1)]
}

calculateEnergy (route, points) {
  let e = 0

  if (route.length === 0) return e
```

```
  for (let i = 0; i < route.length - 1; i++) {
    e += this.metric(points[route[i]], points[route[i + 1]])
  }

  e += this.metric(points[route[route.length - 1]], points[route[0]])

  return e
}

shuffle (a) {
  const result = a.slice()

  for (let i = result.length - 1; i > 0; i--) {
    let j = Math.floor(Math.random() * (i + 1));

    [result[i], result[j]] = [result[j], result[i]]
  }

  return result
}

generateRoute (n) {
  return this.shuffle(Array.from(Array(n).keys()))
}

async simulate (points, initialTemperature, minTemperature, timeout = 0) {
  let result = this.generateRoute(points.length)

  this.dispatch('updateRoute', result)

  let currentEnergy = this.calculateEnergy(result, points)
  this.dispatch('updateEnergy', currentEnergy)

  let t = initialTemperature
  this.dispatch('updateTemperature', t)

  let cycles = 100000

  for (let i = 0; i < cycles; i++) {
    this.dispatch('newIteration', i + 1)

    const stateCandidate = this.generateStateCandidate(result)
    const candidateEnergy = this.calculateEnergy(stateCandidate, points)

    if (candidateEnergy < currentEnergy) {
      currentEnergy = candidateEnergy
      result = stateCandidate

      this.dispatch('updateRoute', result)
      this.dispatch('updateEnergy', currentEnergy)
    } else {
      const p = this.getTransitionProbability(candidateEnergy - currentEnergy, t)
```

```
        if (this.isTransition(p)) {
          currentEnergy = candidateEnergy
          result = stateCandidate

          this.dispatch('updateRoute', result)
          this.dispatch('updateEnergy', currentEnergy)
        }
      }

      t = this.decreaseTemperature(initialTemperature, i)
      this.dispatch('updateTemperature', t)

      if (t <= minTemperature) break

      await delay(timeout)
    }

    return result
  }
}

class DynamicAdaptation extends Dispatcher {
  constructor (worker) {
    super()

    this._worker = worker
  }

  getRandPoints (params) {
    return this._worker.getRandPoints(params)
  }

  calculateEnergy (route, points) {
    return this._worker.calculateEnergy(route, points)
  }

  generateRoute (n) {
    return this._worker.generateRoute(n)
  }

  generateRoutes (cn, n) {
    let result = []

    for (let i = 0; i < cn; i++) {
      result.push(this.generateRoute(n))
    }

    return result
  }

  getBetterRoute (routes, points) {
```

```
  const e = routes.map(i => this.calculateEnergy(i, points))
  const i = e.indexOf(Math.min(...e))
  return routes[i].slice()
 }

 divideArray (route, parts) {
  let groups = []
  let tail = []
  const len = Math.floor(route.length / parts)

  if (route.length % parts !== 0) {
   tail = route.slice(len * parts)
  }

  for (let i = 0; i < parts; i++) {
   groups.push(route.slice(i * len, i * len + len))
  }

  groups[groups.length - 1] = [...groups[groups.length - 1], ...tail]

  return groups
 }

 async simulate (points, periodsCount, solutionsCount, options) {
  let routes = this.generateRoutes(solutionsCount, points.length)
  let result = this.getBetterRoute(routes, points)

  this.dispatch('updateRoute', result)

  let currentEnergy = this.calculateEnergy(result, points)
  this.dispatch('updateEnergy', currentEnergy)

  const parts = this.divideArray(points, periodsCount)

  let r = []
  let index = 0
  let energy = 0

  const pipeRoute = result => {
   this.dispatch('updateRoute', result)
  }
  const pipeEnergy = result => {
   this.dispatch('updateEnergy', energy + result)
  }
  const pipeTemperature = val => {
   this.dispatch('updateTemperature', val)
  }

  this._worker.on('updateRoute', pipeRoute)
  this._worker.on('updateEnergy', pipeEnergy)
  this._worker.on('updateTemperature', pipeTemperature)
```

```
    const p = []
    // for (let i = 0; i < periodsCount; i++) {
    //   this.dispatch('newIteration', i + 1)
    //
    //   p.push(...parts[i])

      // r = await this._worker.simulate(p, options.initialTemperature, options.minTemperature)
      r = await this._worker.simulate(points, options.initialTemperature, options.minTemperature)

      energy = this.calculateEnergy(r, points)

      this.dispatch('updateRoute', r)
      this.dispatch('updateEnergy', energy)

    //   index += parts[i].length
    //   await delay(0)
    // }

    this._worker.off('updateRoute', pipeRoute)
    this._worker.off('updateEnergy', pipeEnergy)
    this._worker.off('updateTemperature', pipeTemperature)

    return r
  }
}

main.js
new Vue({
  el: '#app',
  data: {
    algorithm: 1
  },
  methods: {
    setAlgorithm (val) {
      this.algorithm = val
    }
  }
})
```

Vue.js – JavaScript-фреймворк (https://vuejs.org/)

Index.php

```
<!DOCTYPE html>

<html>
    <head>
        <meta charset="UTF-8">
        <link rel="stylesheet" href="css/bootstrap.min.css">
    </head>
    <body>
        <div class="container">
            <div id="app" class="mt-3 mb-3">
```

```
            <div class="form-check">
                <input class="form-check-input" type="radio"
                    name="exampleRadios" id="exampleRadios1" value="option1" checked
                    @input="setAlgorithm(1)">
                <label class="form-check-label" for="exampleRadios1">
                    Annealing
                </label>
            </div>
            <!--<div class="form-check">-->
                <!--<input class="form-check-input" type="radio"-->
                    <!--name="exampleRadios" id="exampleRadios2" value="option2"-->
                    <!--@input="setAlgorithm(2)">-->
                <!--<label class="form-check-label" for="exampleRadios2">-->
                    <!--Dynamic Adaptation-->
                <!--</label>-->
            <!--</div>-->

            <annealing v-if="algorithm === 1"></annealing>
            <!--<dynamic-adaptation v-if="algorithm === 2"></dynamic-adaptation>-->
        </div>
    </div>
    <script src="js/vue.js"></script>
    <script src="js/DispatcherEvent.js"></script>
    <script src="js/Dispatcher.js"></script>
    <script src="js/lib.js"></script>
    <script src="js/graph.js"></script>
    <script src="js/Annealing.js"></script>
    <script src="js/DynamicAdaptation.js"></script>
    <script src="js/Djibouti.js"></script>
    <script src="js/Luxembourg.js"></script>
    <script src="js/Qatar.js"></script>
    <script src="js/main.js"></script>
</body>
</html>
```